WARPM Framework for Advanced Plasma Model Simulations on Many-Core Architectures

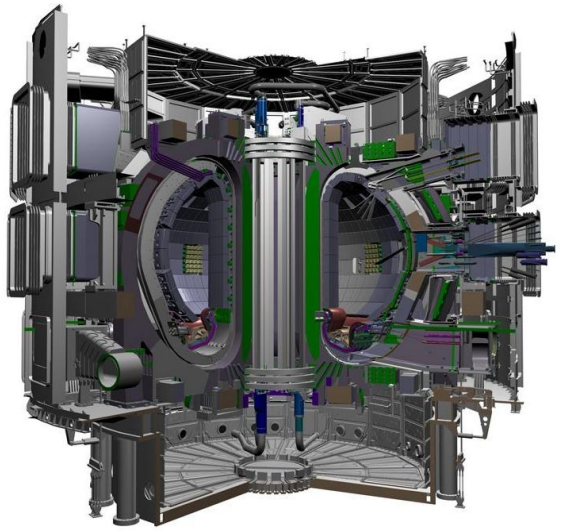
Noah Reddell

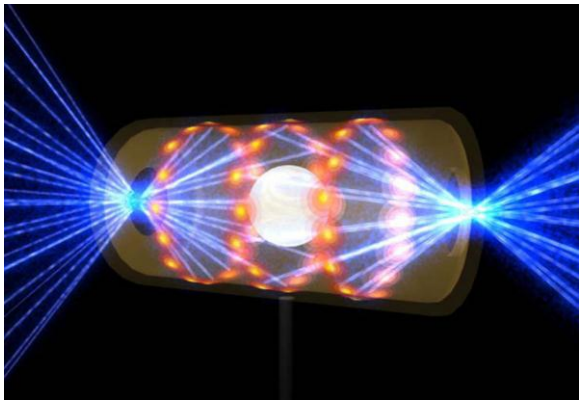
University of Washington

DOE CSGF

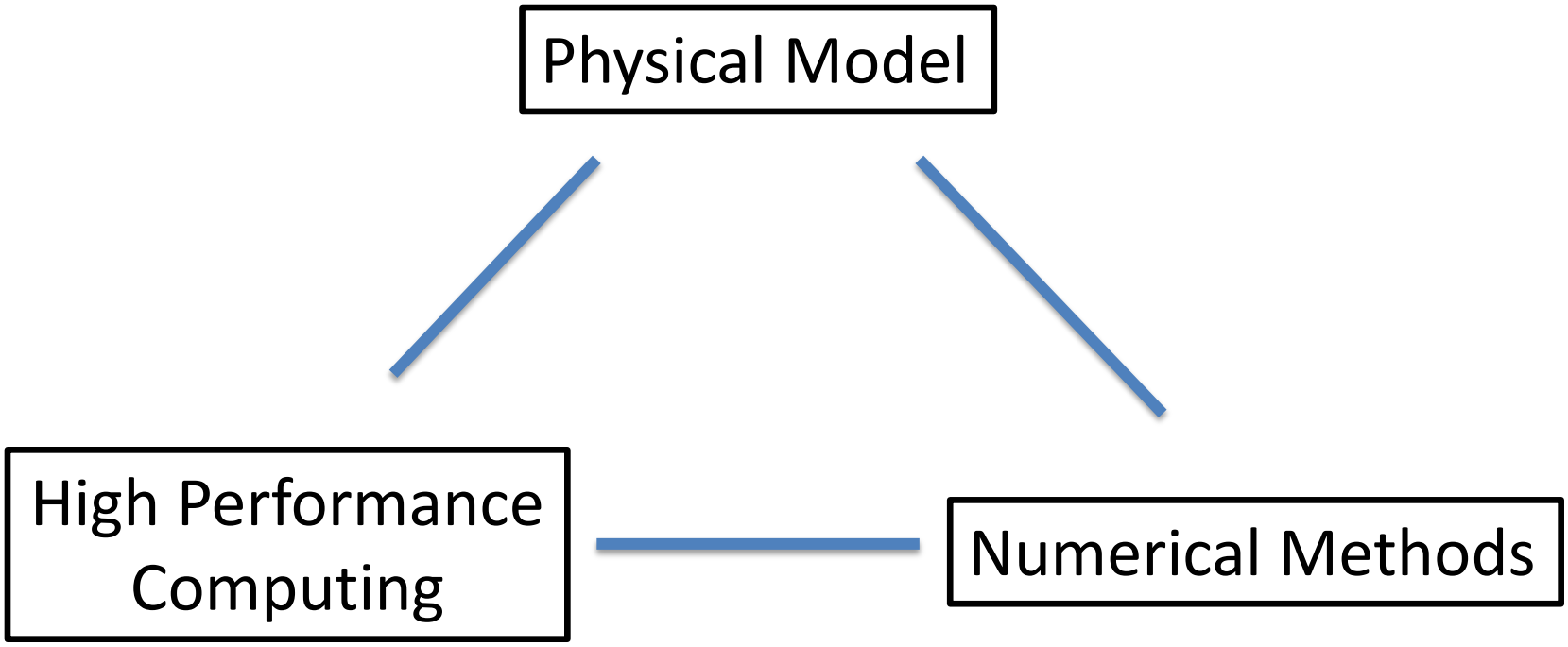
Next-Generation Plasma Simulation

- Goal: model multiple species (ions, electrons, neutrals, etc.) as separate explicit fluids that interact through both surface and body forces including coupling to electromagnetic fields.
- Allow for dynamic regime where plasma is not in local thermodynamic equilibrium through **full kinetic representation.**
- Applications: fusion innovative confinement concepts, plasma actuators for supersonic craft, industrial apps., ICF, fundamental physics.



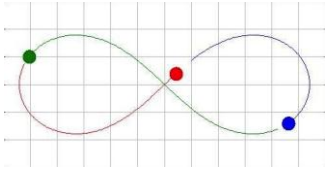


A Role for Computational Science



Hierarchy of common plasma models

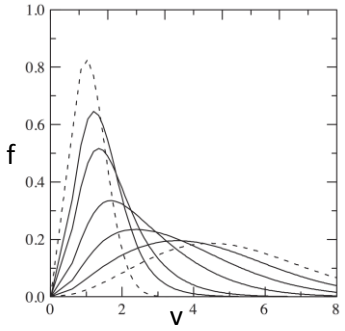
- N-body Model



$$\frac{d\vec{v}_i}{dt} = \frac{q_i}{m_i}\left(\vec{E} + \vec{v}_i \times \vec{B}\right) + \sum_{i \neq j} \left(\frac{d\vec{v}_i}{dt}\right)_{\text{coll.}}^{\text{2-body}} \delta(\vec{r}_j - \vec{r}_i) + \cdots$$

$$\frac{d\vec{r}_i}{dt} = \vec{v}_i$$

- Kinetic Model



$$\frac{\partial f_\alpha}{\partial t} + \vec{v} \cdot \frac{\partial f_\alpha}{\partial \vec{x}} + \frac{q_\alpha}{m_\alpha}\left(\vec{E} + \vec{v} \times \vec{B}\right) \cdot \frac{\partial f_\alpha}{\partial \vec{v}} = \left(\frac{\partial f_\alpha}{\partial t}\right)_{\text{coll.}}$$

Hierarchy of common plasma models

- Fluid Model
  - Reduce the detailed information contained in the distribution function through moments.
  - Eg. 0th moment gives number density

$$M_{\alpha,n}(\vec{x}) = \int (\vec{v})^n f_\alpha(\vec{x}, \vec{v}) d\vec{v}$$

  - 0th, 1st, 2nd moments sufficient to describe Maxwellian distribution of local thermodynamic equilibrium.
  - Physical closure rule always needed to truncate the series. (eg. Ideal gas law)

Hierarchy of common plasma models

- Magnetohydrodynamics (MHD)
  - Starting from two-fluid (e^-, i^+) model, asymptotic approximations are made:

$$\epsilon_0 \to 0, \qquad c \to \infty, \qquad m_e \to 0$$

  - 'single' fluid variable evolution of
    - mass density
    - velocity
    - pressure
    - current density

Continuum kinetic model

- Represent physics of the most complex High Energy Density Plasmas and transient conditions
- Conserves mass, momentum, and energy
- Time advance limited by speed of light velocity extrema
  - Existing alternatives include Particle-In-Cell and Semi-Lagrangian methods.
- Up to 6D Phase Space!

Vlasov-Maxwell system for plasma

$$\frac{\partial f_s}{\partial t} + \mathbf{v} \cdot \frac{\partial f_s}{\partial \mathbf{x}} + \frac{q_s}{m_s}(\mathbf{E} + \mathbf{v} \times \mathbf{B}) \cdot \frac{\partial f_s}{\partial \mathbf{v}} = 0,$$

$$\begin{aligned} -\frac{1}{c^2}\frac{\partial \mathbf{E}}{\partial t} + \nabla \times \mathbf{B} &= \mu_0 \mathbf{J}, \\ \frac{\partial \mathbf{B}}{\partial t} + \nabla \times \mathbf{E} &= 0, \\ \nabla \cdot \mathbf{E} &= \frac{\rho}{\varepsilon_0}, \\ \nabla \cdot \mathbf{B} &= 0. \end{aligned}$$

with

$$\rho(\mathbf{x}, t) = \sum_s q_s \int f_s(\mathbf{x}, \mathbf{v}, t)\, d\mathbf{v}, \qquad \mathbf{J}(\mathbf{x}, t) = \sum_s q_s \int f_s(\mathbf{x}, \mathbf{v}, t)\mathbf{v}\, d\mathbf{v}.$$

Vlasov-Maxwell system for plasma

$$\frac{\partial f_s}{\partial t} + \mathbf{v} \cdot \frac{\partial f_s}{\partial \mathbf{x}} + \frac{q_s}{m_s}(\mathbf{E} + \mathbf{v} \times \mathbf{B}) \cdot \frac{\partial f_s}{\partial \mathbf{v}} = \left.\frac{\mathrm{d} f_s(\mathbf{x}, \mathbf{v}, t)}{\mathrm{d} t}\right|_{collision}$$

Or consider collisions.

$$\begin{aligned} -\frac{1}{c^2}\frac{\partial \mathbf{E}}{\partial t} + \nabla \times \mathbf{B} &= \mu_0 \mathbf{J}, \\ \frac{\partial \mathbf{B}}{\partial t} + \nabla \times \mathbf{E} &= 0, \\ \nabla \cdot \mathbf{E} &= \frac{\rho}{\varepsilon_0}, \\ \nabla \cdot \mathbf{B} &= 0. \end{aligned}$$

with

$$\rho(\mathbf{x}, t) = \sum_s q_s \int f_s(\mathbf{x}, \mathbf{v}, t)\, d\mathbf{v}, \qquad \mathbf{J}(\mathbf{x}, t) = \sum_s q_s \int f_s(\mathbf{x}, \mathbf{v}, t)\mathbf{v}\, d\mathbf{v}.$$

Vlasov-Poisson validation

- The Vlasov-Poisson model for plasma considers a **single** charged particle species without collisions, coupled to macroscopic **electrostatic** field forces.

- Probability distribution evolution as before.

- Electric field is evaluated globally, considering a fixed background species with net zero charge.

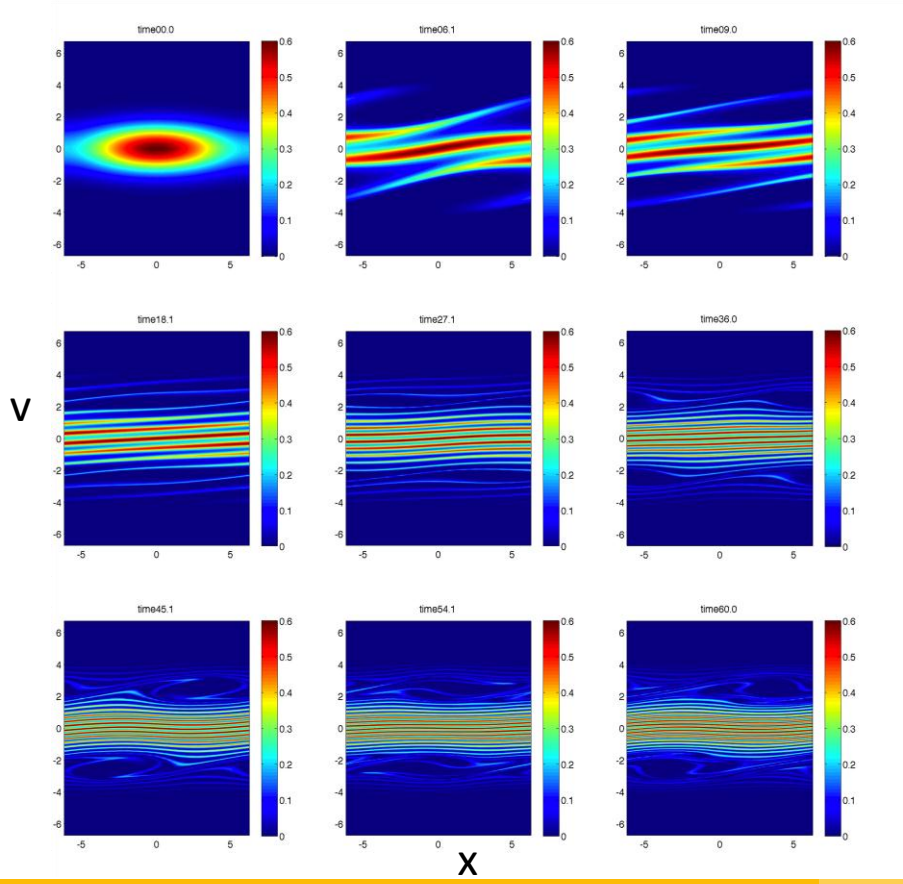
$$\nabla \cdot \vec{E} = \frac{\sigma}{\epsilon_0},$$
$$\vec{E} = -\nabla\psi,$$
$$\nabla^2\psi = -\frac{\sigma}{\epsilon_0}.$$

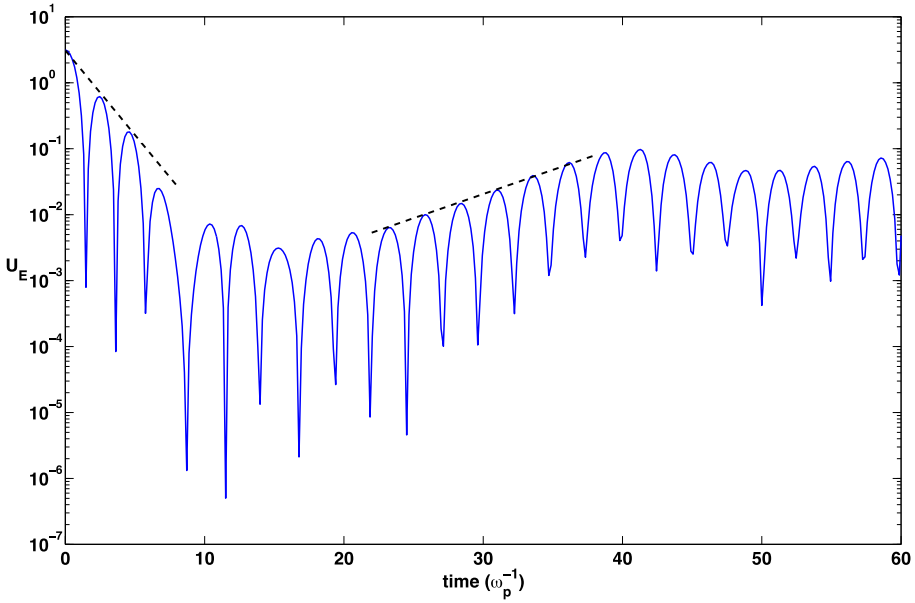
- We solve the model in on a fixed Eulerian grid using discontinuous Galerkin method with arbitrary order of accuracy. Accuracy is controlled by grid resolution and order of Legendre polynomial basis set.

- Classic results are reproduced utilizing normalization q/m=1, such that time is in units of the plasma period $(\omega_p)^{-1}$.

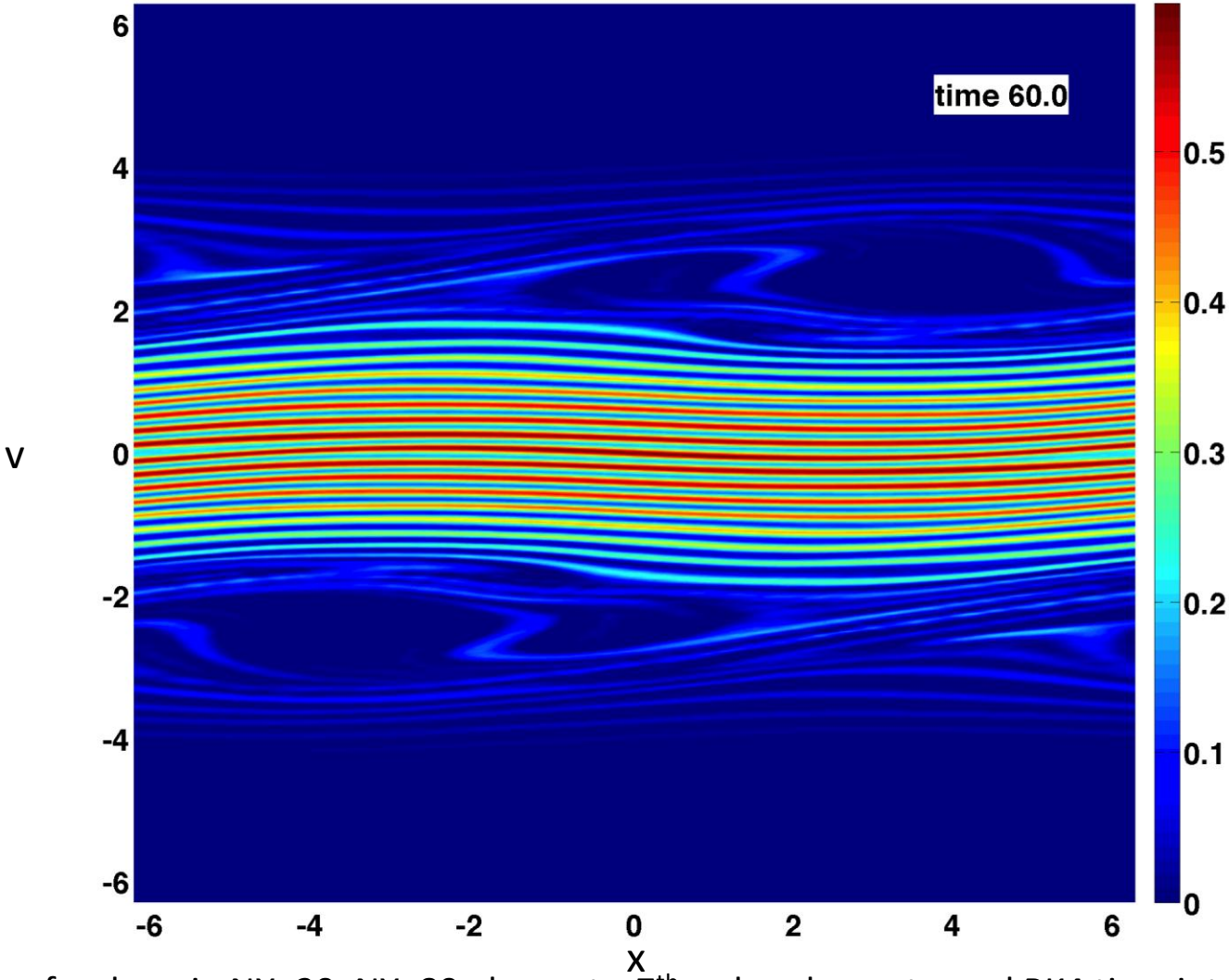
Strong Landau Damping

- Starting with a strongly perturbed Maxwellian distribution, non-linear mode growth dominates the long term dynamics.
- Results demonstrate fine scale striation in phase space realized through DG, and agreement with expected phase space evolution and growth rates of electric field energy (γ_1=-0.5904 , γ_2= 0.1688).





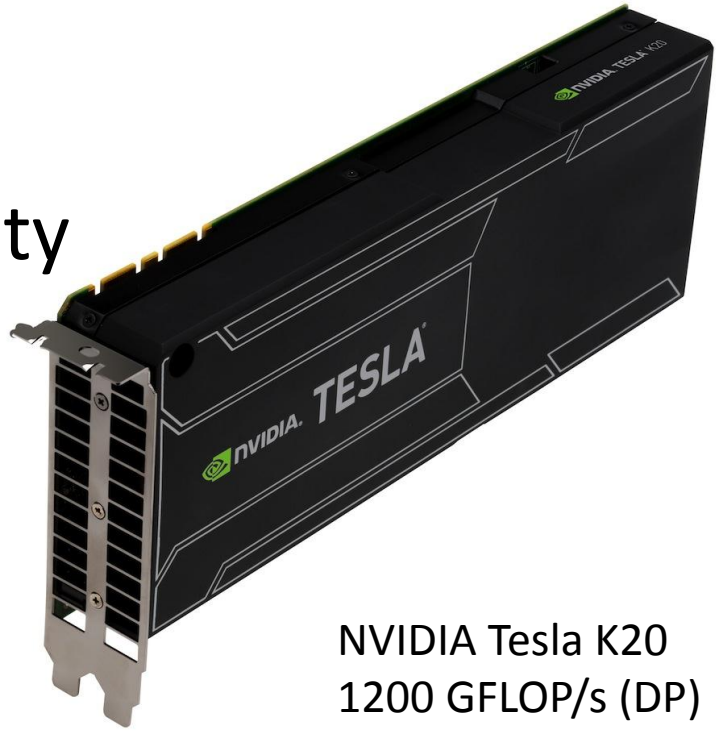
Strong Landau Damping



Results shown for domain NX=20, NY=80 elements, 7th order elements, and RK4 time integration.

Contemporary GPU architecture

- High degree of parallelism
- Multiple memory hierarchies
- Lowest energy per FLOP
- Highest FLOP/s physical density
- New programming models required to achieve good performance



NVIDIA Tesla K20
1200 GFLOP/s (DP)
2496 cores

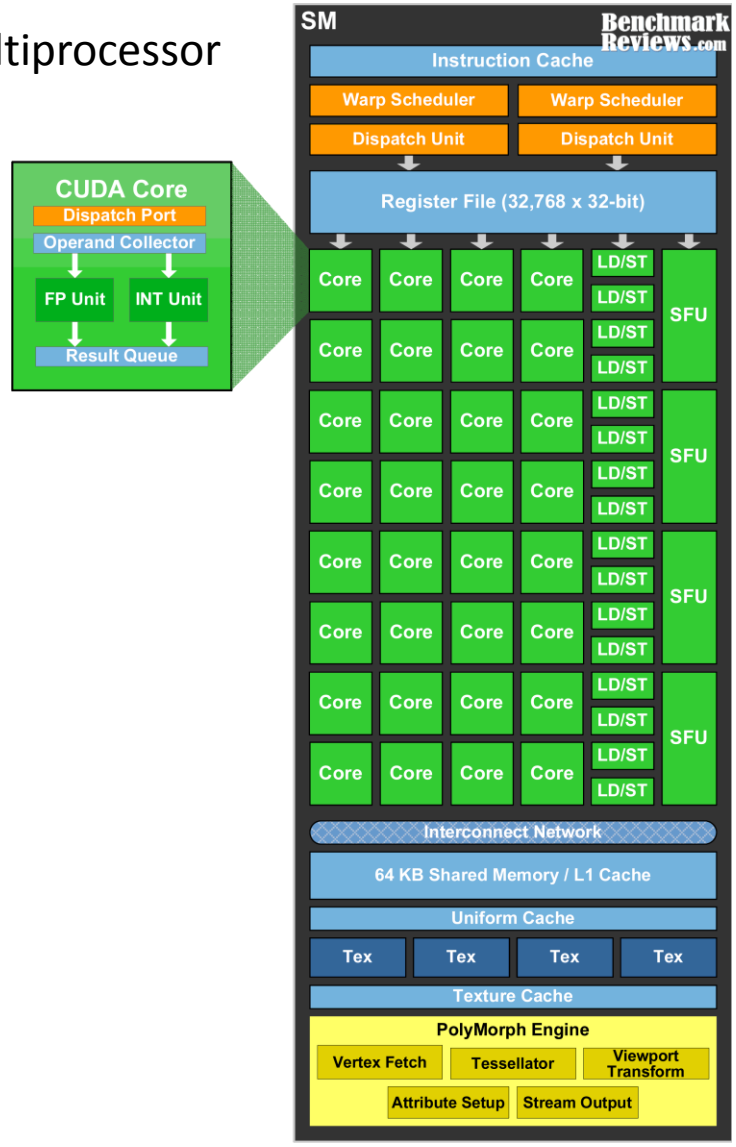
Contemporary GPU architecture

- 512 - 2,496 cores or active threads
- 4kB private memory per core (Fermi)
- 32 - 192 cores comprising workgroup
  - lock-step simultaneous instruction execution
  - Shared load/store units
  - Up to 64 kB shared local memory among 32 cores (Fermi)
  - Can only synchronize within workgroup

Coordinated Lightweight Cores

NVIDIA Streaming Multiprocessor

-> one ‘workgroup’



Reduced data movement is critical

Example:

- PCIe 2.0 x16 – 8 GB/s host to GPU
- Using NVIDIA Fermi M2050 specs:
- Peak performance: 515 GFLOP/s DP
- 515 GFLOP/s ÷ 8 GB/s × 8 bytes/double)
  - **515 floating point operations** needed per operand transferred between GPU and host to hide PCIe bottleneck
- GPU Memory bandwidth: 148 GB/s to cores
  - **28 floating point operations** needed per operand transferred between GPU GRAM and core to hide memory interface bottleneck

Data movement severely affects power

- Energy requirements for one FLOP around 50 pJ for FMA with operands held locally in registers or L1 cache.
- But moving data and other overhead accounts for about 1k-10k pJ per FLOP.
  - In one study 1.3k pJ for local DRAM, 14k pJ for distributed (MPI) memory access.[1]

1. Kogge, 2011 Hardware Evolution Trends of Exascale Computing.

OpenCL Programming Model

- Open and royalty-free standard for multi-core and many-core programming supported by dozens of HPC companies.

- …but implemented by ~~few~~ a growing base
  - NVIDIA CUDA SDK on Linux
  - AMD Streaming SDK on Linux
  - Apple OS X (AMD+NVIDIA GPU, Intel CPU)
  - Intel OpenCL SDK
  - IBM OpenCL Development (POWER7 CPU)
- Based on C99 with additions and omissions
- Not very abstracted. Developer must explicitly consider data locality and hardware specifics
- Similar to NVIDIA's CUDA language

OpenCL kernels

Single-threaded C version:

```
void fill_tiles( const float* a,
                 const float* b,
               restrict float* c,
                 int N, int M)
{
  // loop over all work
  for( int row=0; row< N; row++)
    for( int col=0; col< M; col++)
    {
      float aTile[TILE_DIM_Y][TILE_DIM_X];
      float bTile[TILE_DIM_Y][TILE_DIM_X];
      for( int y=0; y< TILE_DIM_Y; y++)
        for( int x=0; x< TILE_DIM_X; x++)
        {
          aTile[y][x] = a[row*M + col];
          bTile[y][x] = b[row*M + col];
        }
        c[row*N+col] =
                    aTile[x][y]*bTile[y][x];
     }
}
```

OpenCL version:

```
kernel void fill_tiles( global const float*
a,
                        global const float* b,
                      global restrict float* c)
{
  // find our coordinates in the grid
  int row = get_global_id(1);
  int col = get_global_id(0);

  //allocate memory shared among the
workgroup
  local float aTile[TILE_DIM_Y][TILE_DIM_X];
  local float bTile[TILE_DIM_Y][TILE_DIM_X];

  // define the coordinates of this workitem
  // thread in the 2D tile
  int y = get_local_id(1);
  int x = get_local_id(0);

  aTile[y][x] = a[row*M + col];
  bTile[y][x] = b[row*M + col];
  barrier(CLK_LOCAL_MEM_FENCE);

  //Note the change in tile location in
bTile!
  c[row*N + col] = aTile[x][y] * bTile[y][x];
```

WARPM: A new simulation code

- Computing advances have brought simulation of multi-fluid and kinetic models within reach
- Hardware architecture features must be incorporated at a low-level
- Modern high-order numerical methods now exist for accurate long-time integration of PDE
- The plasma physics community could really use a better tool

WARPM: Design goals

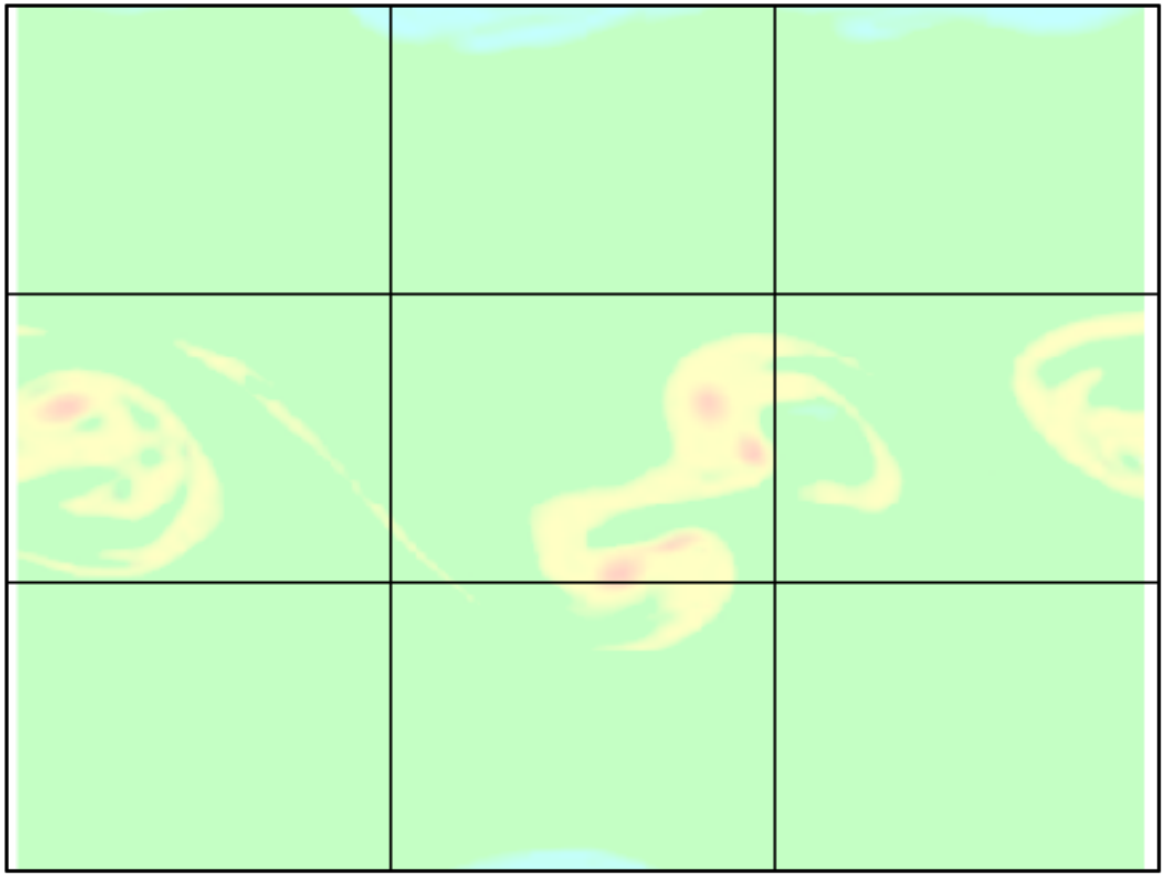
- Desire a code flexible for a wide class of problems and based on enduring standards and libraries.
  - C++, OpenCL, boost, HDF5, GlobalArrays
- Design for many-core architectures
- Reduce memory movement
- Achieve excellent weak scaling
- Be useful to others in the community
- **WARPM code has been developed to meet these goals.**

WARPM Framework

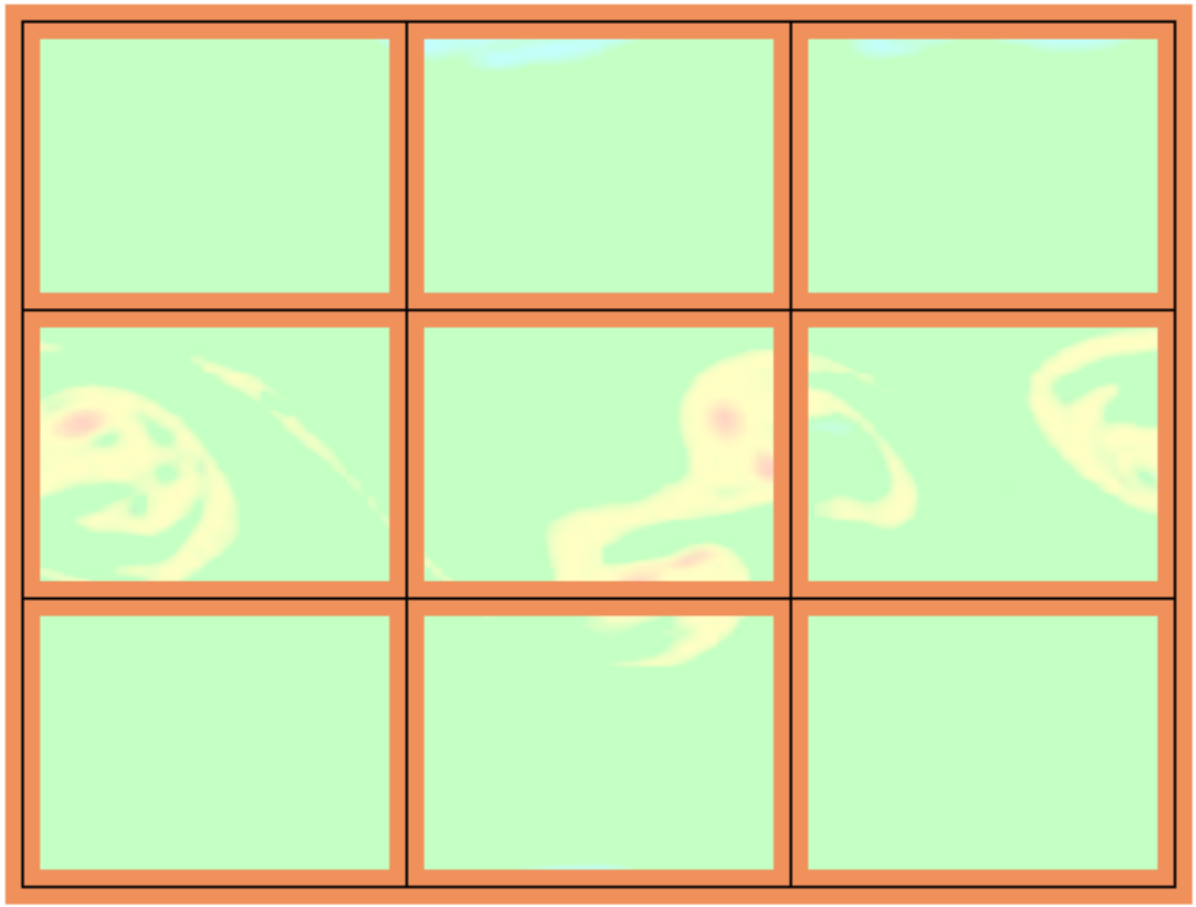
- Supports local/explicit numerical methods for hyperbolic systems
- '*M*' for many-core
- Three levels of parallelism
  - MPI (communication between nodes)
  - Threads (task parallelism)
  - OpenCL (data parallelism)
- Natural support for heterogeneous computing on HPC clusters

Multi-level domain decomposition

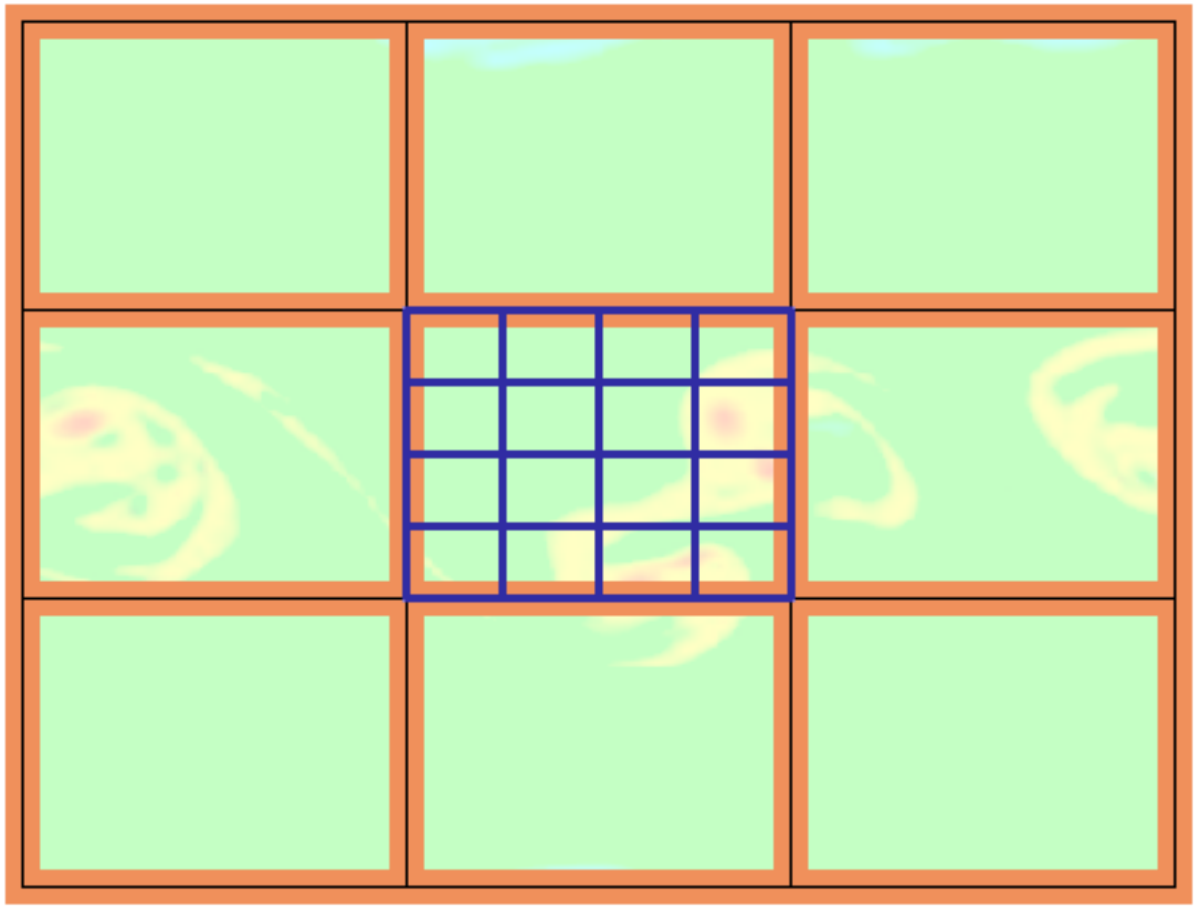
Multi-level domain decomposition



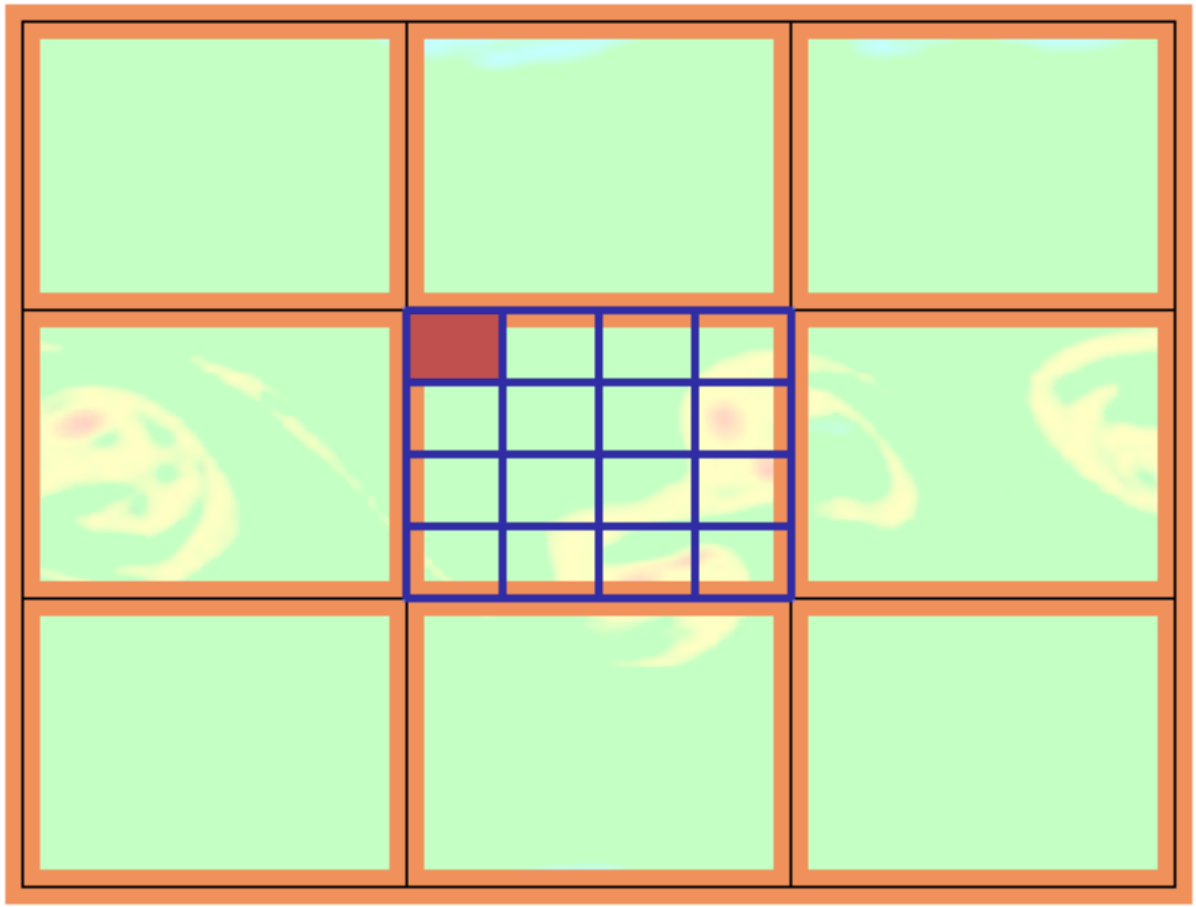
Multi-level domain decomposition



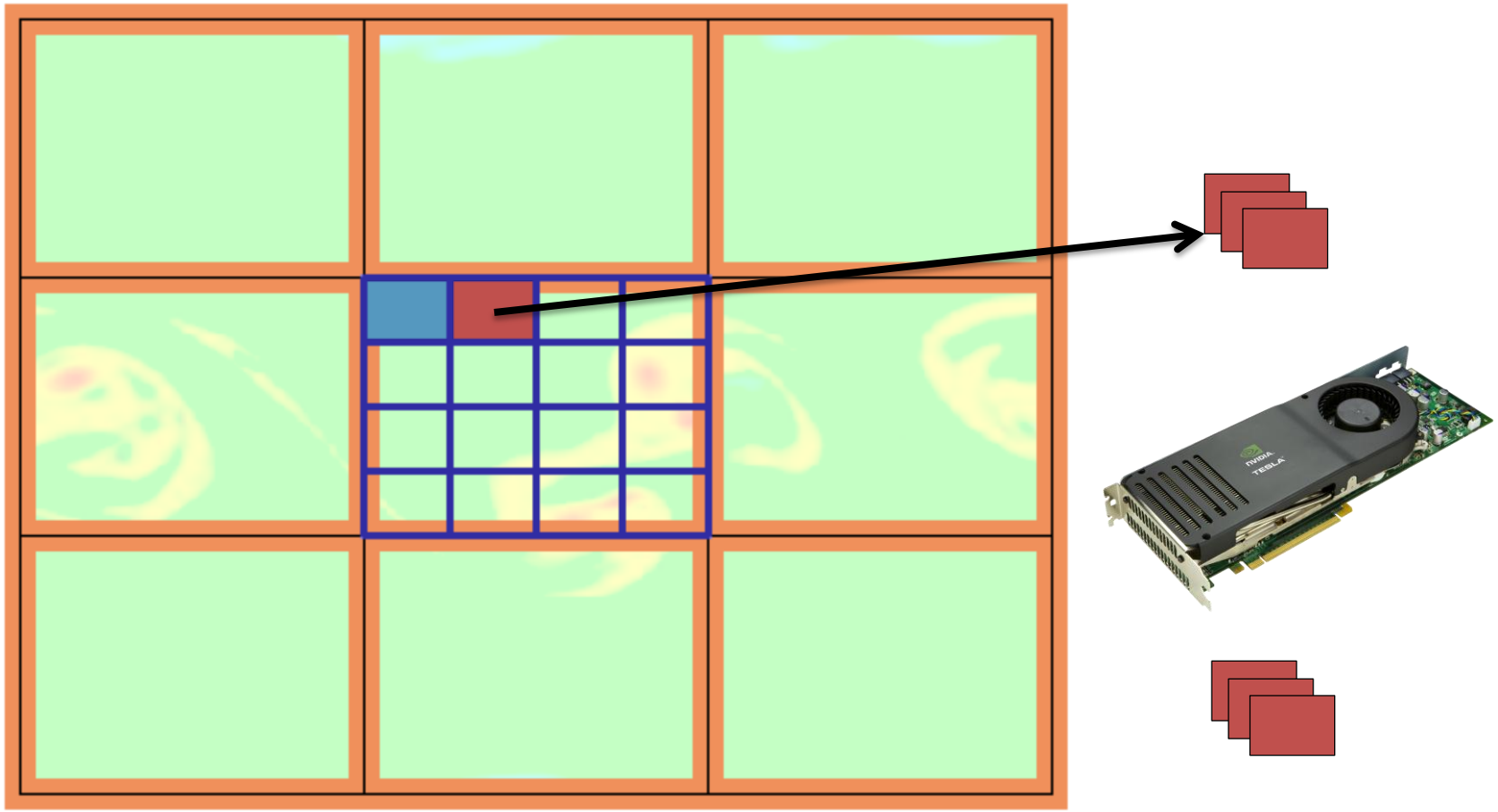
Multi-level domain decomposition



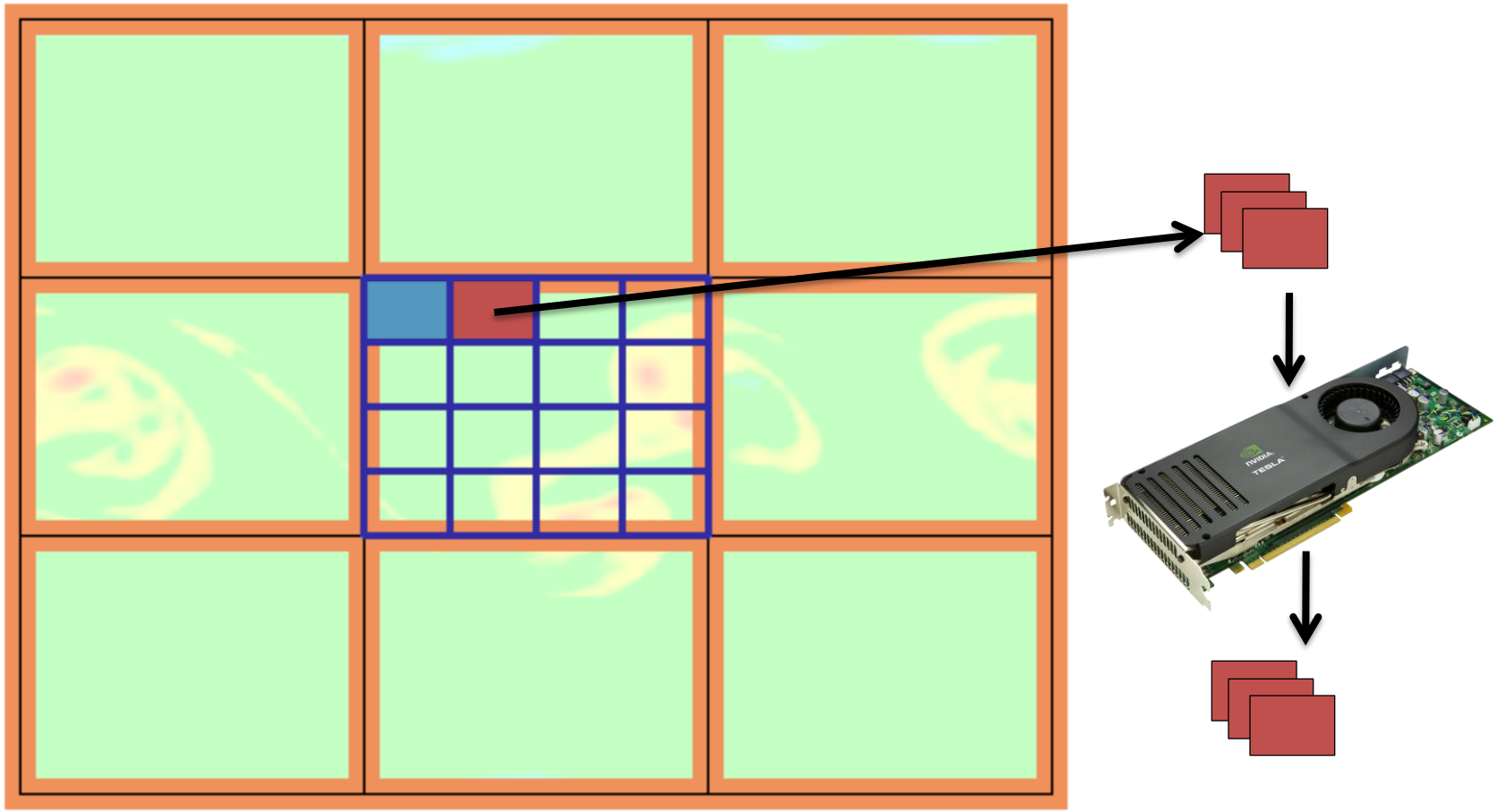
Multi-level domain decomposition



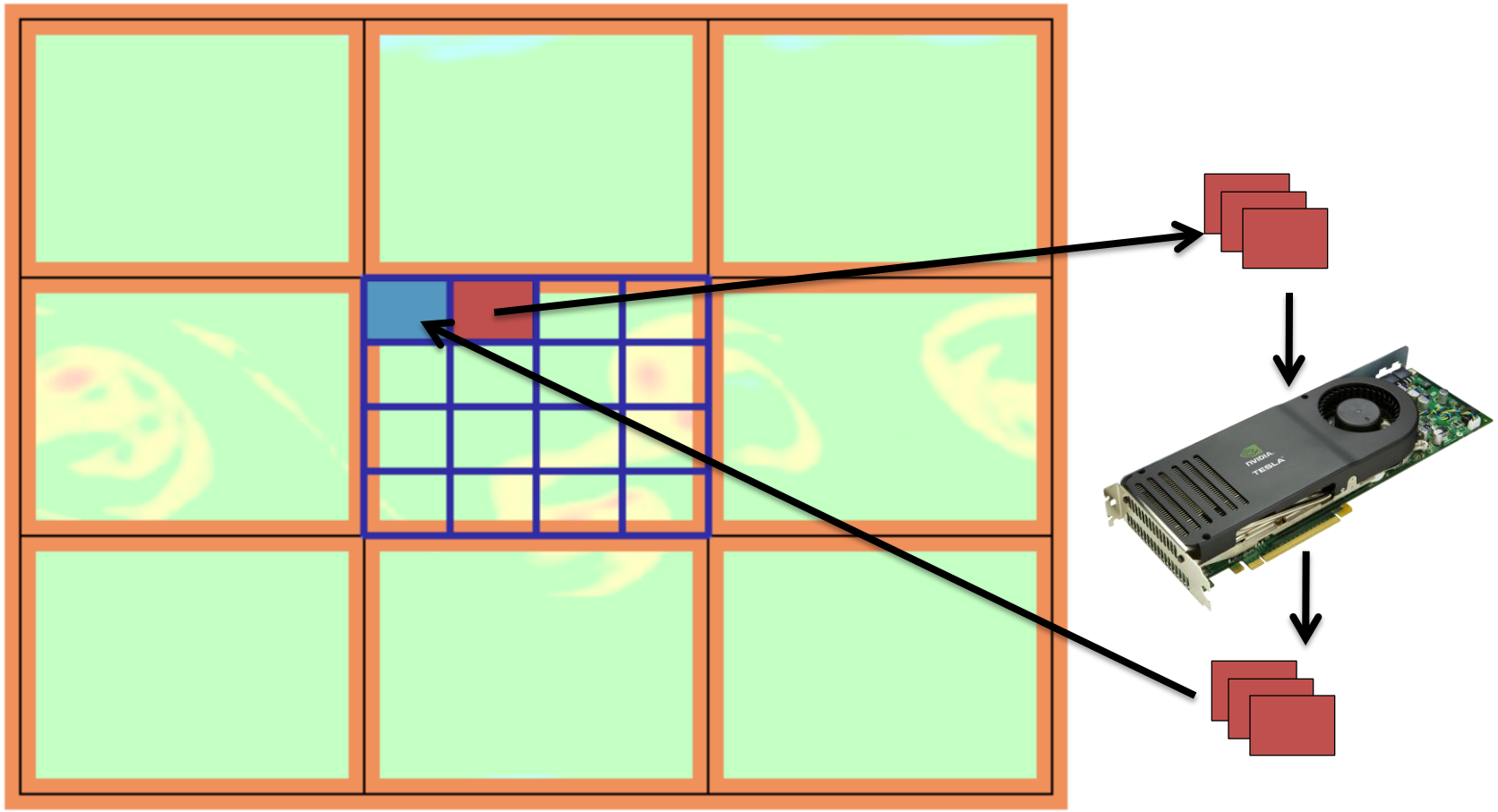
Multi-level domain decomposition



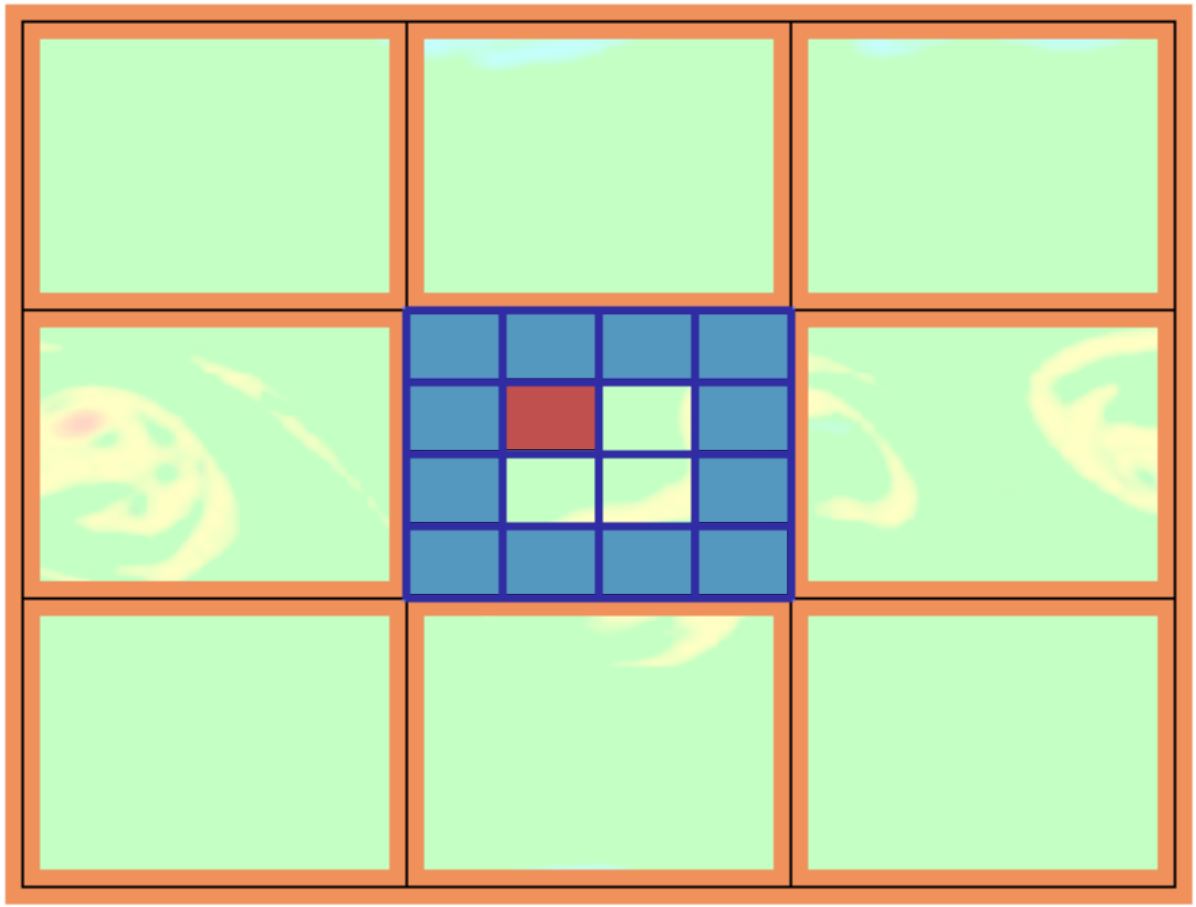
Multi-level domain decomposition

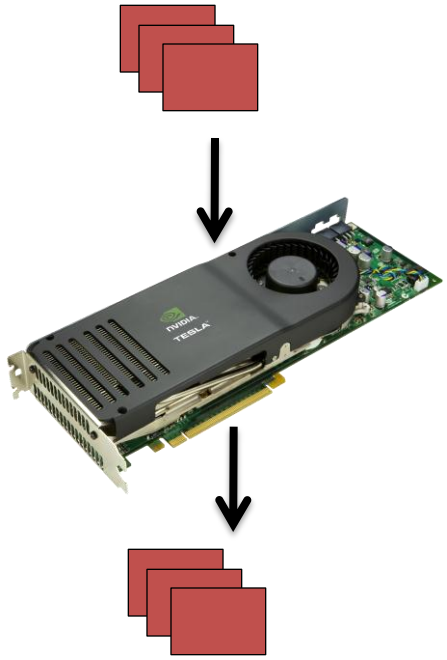


Multi-level domain decomposition

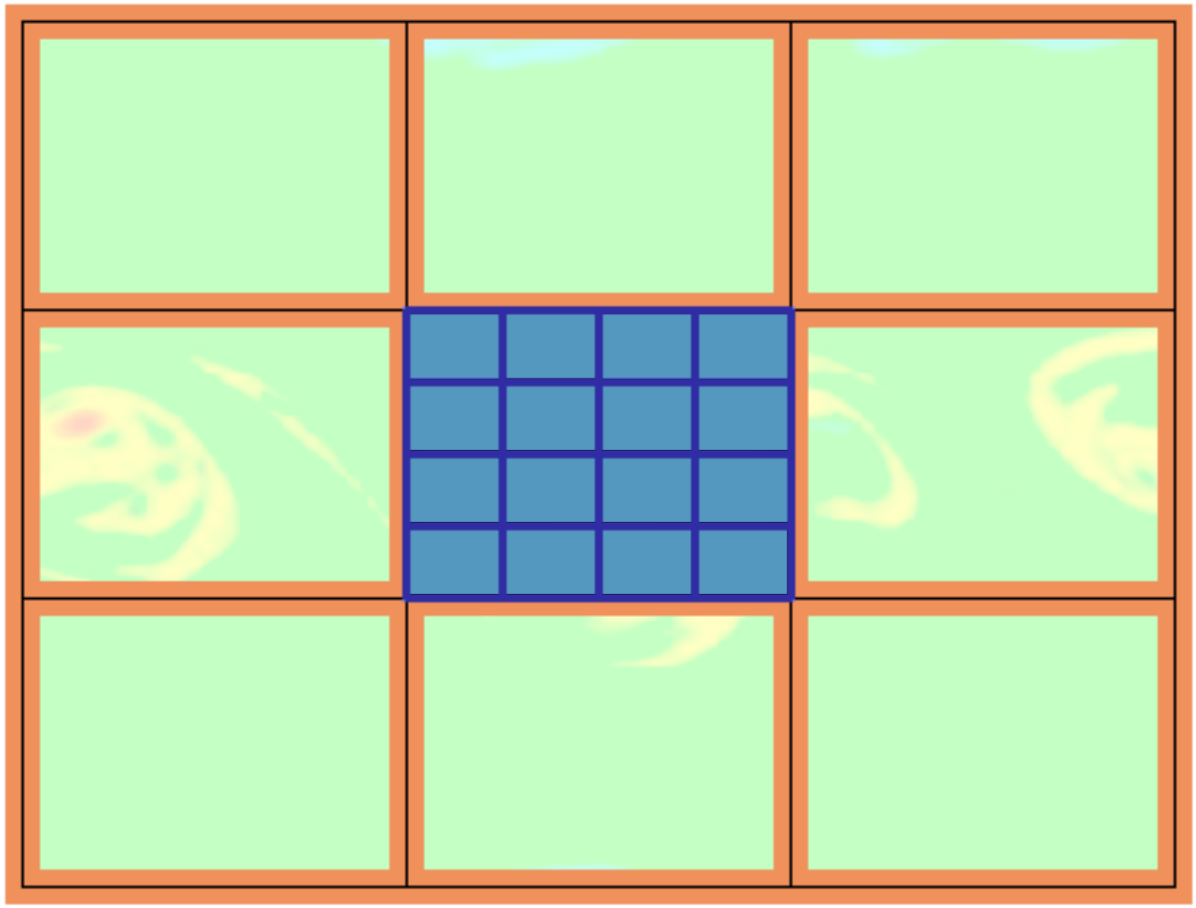


Multi-level domain decomposition





Multi-level domain decomposition

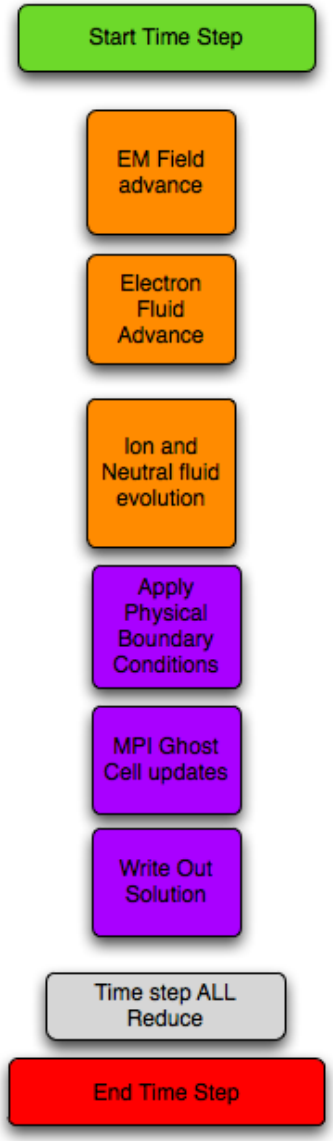


Previous Computing Model

Transformed our previous plasma simulation code WARPX which relies solely on MPI for parallelism to one that makes better use of modern architectures.

Starting point: 1 MPI process per core, all memory sharing through MPI Send/Receive, all steps execute in series.

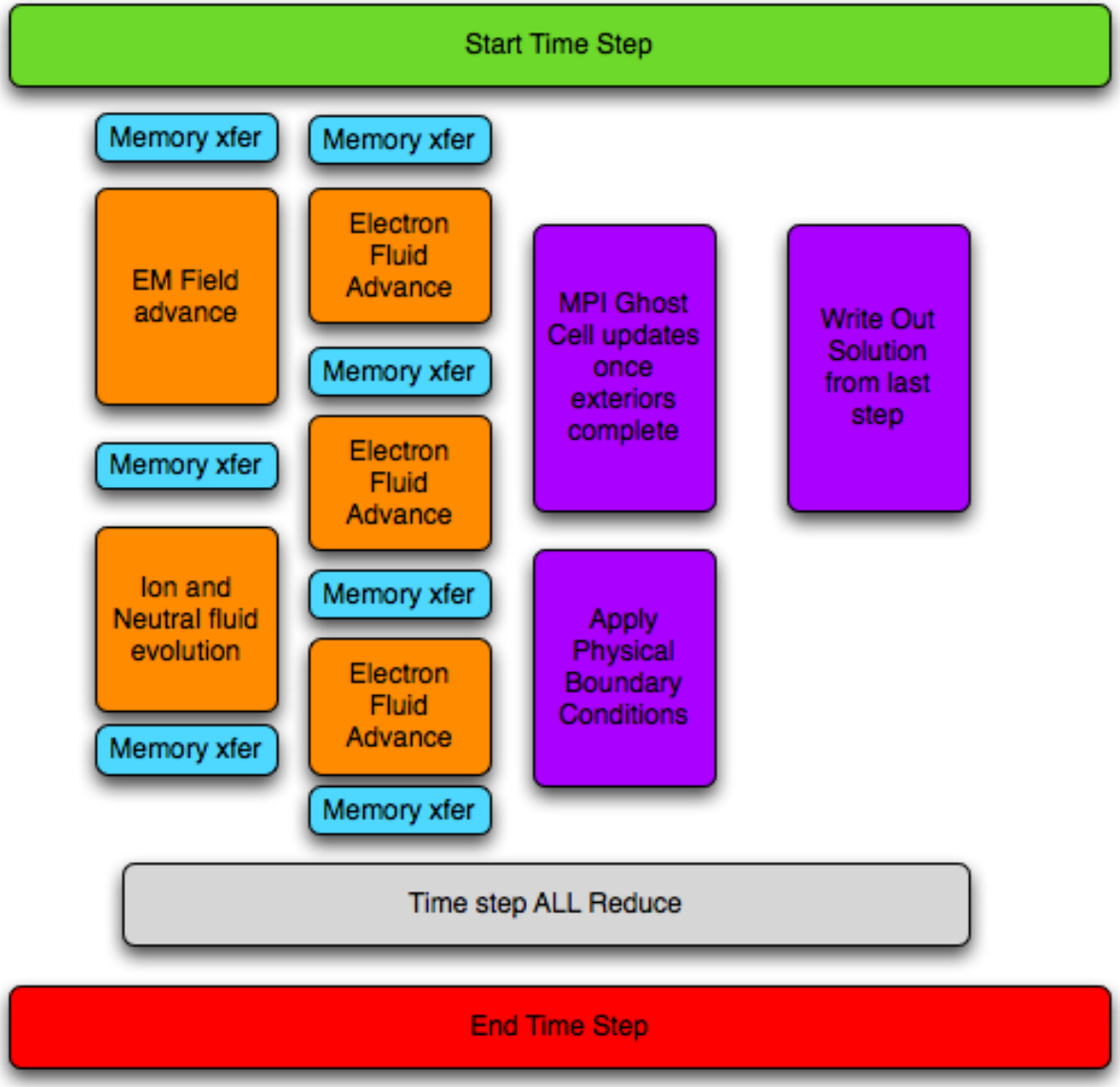
Common story for many HPC codes.



Task and data parallelism

Mix of GPU and CPU computing. OpenCL, pthreads, and MPI parallel execution models combined.

First attempt at OpenCL implementation uses multiple kernels and thus more GPU global memory transfer than necessary.

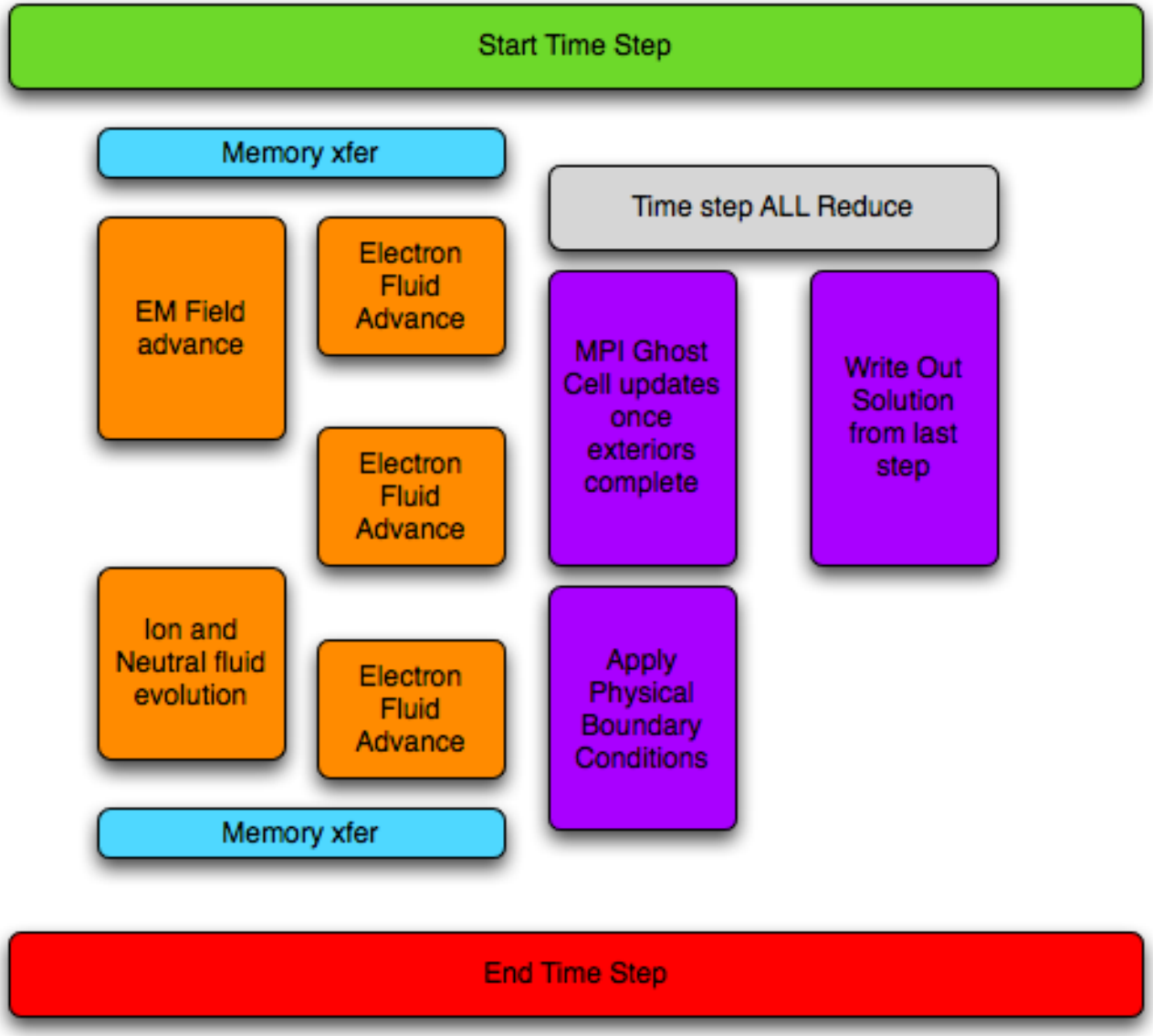


Task and data parallelism

Revised OpenCL implementation dynamically assembles source code from modules into a single consolidated kernel.

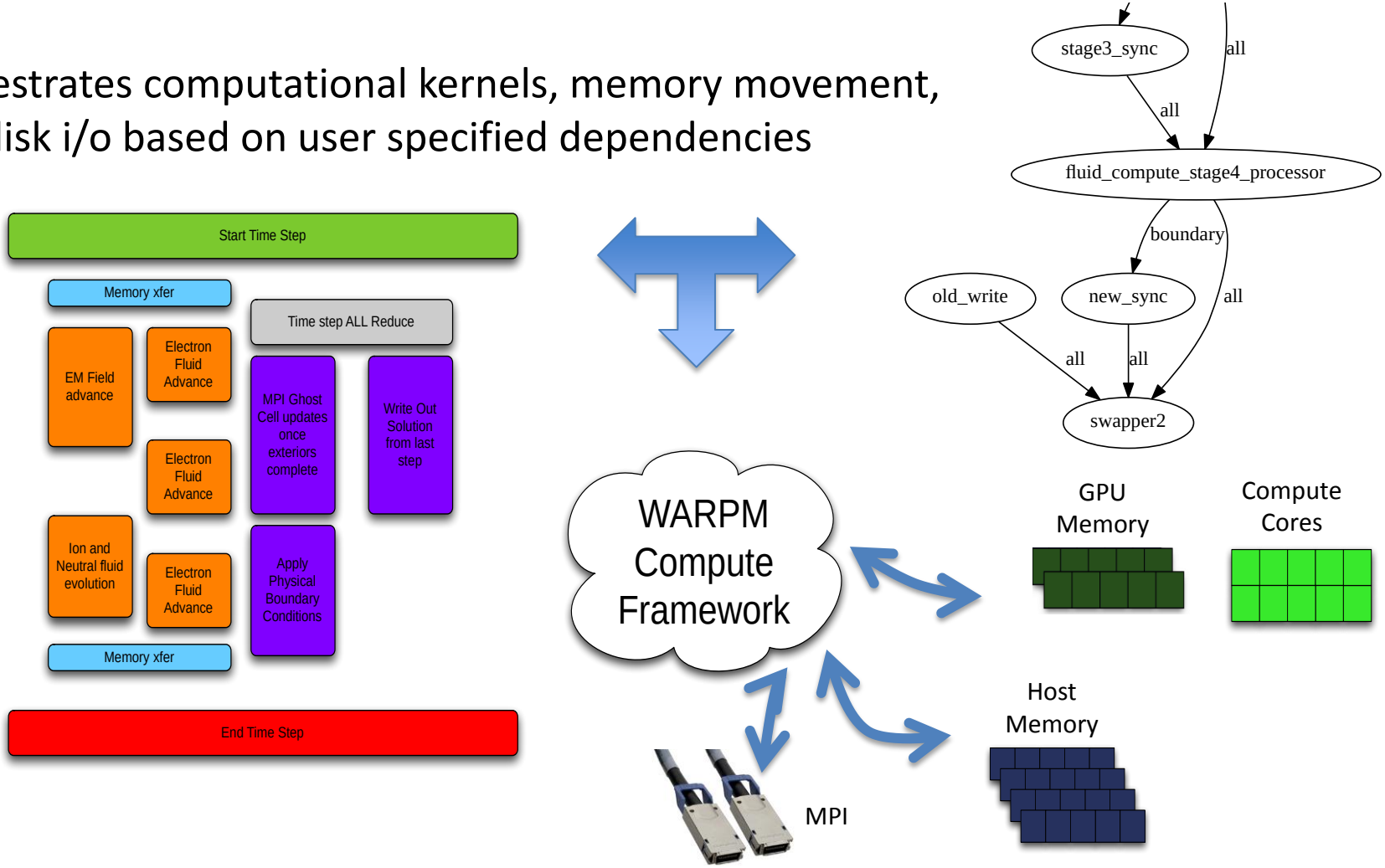
Reduces core to GPU RAM transfers

Single compilation block maximizes compiler optimizations.



WARPM Framework

Orchestrates computational kernels, memory movement, and disk i/o based on user specified dependencies



Users supply the computational model and evaluation sequence.

Summary

- WARPM has proven to be an effective new scientific computing framework that is well situated for emerging computing architectures. Development continues amongst a small team.
- Three major technologies incorporated:
  - Dynamic OpenCL source assembled from modules
  - Periphery-first computation with task-parallel communication
  - Multi-level work domain decomposition that complements hardware model
- WARPM provides a framework that can be readily extended by users to solve their own computational models.

Upcoming Work

- Kernel optimization efforts
- Weak scaling studies on ?
- Vlasov-Maxwell model implementation in WARPM
- Physical Investigations

Thank you

- All of this is possible with support from my advisor, Prof. Uri Shumlak
- and the DOE Computational Science Graduate Research Fellowship.